SCALEABLE GAP INSERTION IN A DATA LINK

BACKGROUND OF THE INVENTION

A. Field of the Invention

[0001] The present invention relates generally to high-speed data communications and, more particularly, to the framing of data transmitted over point-to-point links.

B. Description of Related Art

[0002] In certain high-speed data transmission links, data is transmitted as a continuous stream. For example, in the packet-over-SONET (POS) optical protocol, packets may be transmitted as a continuous data stream of bytes in which a number of consecutive bytes may comprise a larger unit of data such as a packet. Because no control information is transmitted external to the data stream, the receiving device delineates between packets in the stream based solely on the content of the stream.

[0003] In one conventional technique for framing such data streams, the receiving device delineates between packet boundaries by looking for predefined control bytes that are inserted within the data stream. Occasionally, actual content data may coincidentally have the same value as the control information. In this situation, the content data is preceded by an "escape byte" that lets the receiving device know that the next byte is content data.

[0004] Figs. 1A and 1B are configurations of an exemplary stream of bytes before and after insertion of escape bytes. In Fig. 1A, a stream of four bytes 100 includes "h10", "h7D", "h08", and "h7E", where "h" indicates that the numbers are base 16 (hexadecimal) numbers. In this example, assume that "h7D" is used as the predetermined escape byte and "h7E" is a predetermined control byte that indicates the end of a packet.

[0005] The second and fourth bytes of packet stream 100 contain data that is identical to the escape byte "h7D" and the control byte "h7E", respectively, and should therefore be escaped. Fig. 1B illustrates stream 100 after having "h7D" and "h7E" replaced. The second byte, "h7D," is replaced with the escape character "h7D" followed by the original byte exclusive ORed with "h20." The fourth byte, "h7E," is replaced with the escape character "h7D" followed by the original byte exclusive ORed with "h20." In this manner, the original four byte stream is expanded to a six byte stream.

[0006] The exclusive OR operation is performed so that the transmitted byte will no longer be a control character. The receiving device receives the escaped data stream of Fig. 1B, recognizes the "h7D" escape characters, removes the "h7D" escape characters, and XORs the character following the escape character with "h20" to thus recover the original data stream 100.

[0007] In high-bandwidth situations, a number of byte streams may be received simultaneously to create a multi-byte datapath. For example, in a

three byte datapath, the bytes of a packet may be transmitted and received simultaneously in three byte blocks.

[0008] An escaping circuit for inserting escape characters in a multi-byte datapath, in one implementation, includes a number of pipelined stages. The first stage surrounds each received byte with blank bytes. A second stage of the pipeline then examines the data stream and, when appropriate, inserts escape characters in the blank bytes. A third stage is a "bytepacker" that compresses the data stream to remove any remaining blank bytes. One problem associated with this implementation is that as the datapaths become wider (e.g., a 32 byte datapath), the logic required to implement the pipeline stages, and in particular, the bytepacking stage, increases to a potentially untenably complicated level.

[0009] Thus, there is a need in the art to implement an efficient escaping circuit that easily scales as the size of the datapath increases.

SUMMARY OF THE INVENTION

[0010] Systems and methods consistent with the principles of the invention, among other things, efficiently insert escape characters in a multi-byte data path.

[0011] One aspect of the invention is directed to a device for inserting escape characters into a multi-byte wide data stream. The device includes a gap insertion component that receives blocks of data of the multi-byte wide data stream and rearranges the bytes of a block of the data stream by

inserting gaps into the block at locations adjacent to bytes that have a value coincident with predefined control characters. Additionally, an escape character inserter inserts escape characters in each of the gaps inserted by the gap insertion component.

[0012] A second aspect of the invention is directed to a method for inserting gaps in a data stream that includes a number of data elements transmitted as parallel blocks of data. The method includes generating a sum for each data element of an active block based on whether the data element has a value equivalent to a control character and on the sum of more significant data elements in the active block. Additionally, the method includes comparing the sum associated with each of the data elements of the active block with a total number of data elements in the active block and creating a bit mask based on the comparison. Finally, the method includes shifting a number of data elements out of the active block based on the bit mask.

[0013] In yet another aspect of the invention, a network device includes a switch fabric and a number of processing elements. The processing elements communicate with one another over the switch fabric and receive external information formatted as data items in a multi-item wide data stream. Moreover, at least one interface connects to each of the processing elements. The interface formats the data streams before transmitting the data stream to others of the processing elements. The interface includes a gap insertion component that rearranges blocks of the data items by inserting gaps in the blocks of the data items at locations adjacent to the data items that have a value

coincident with a predefined control character, and an escape character inserter that inserts escape characters in each of the gaps inserted by the gap insertion component.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, explain the invention. In the drawings,

[0015] Figs. 1A and 1B are configurations of an exemplary stream of bytes before and after insertion of escape bytes;

[0016] Fig. 2 is a block diagram illustrating an exemplary routing system in which systems and methods consistent with the principles of the invention may be implemented;

[0017] Fig. 3 is a detailed block diagram illustrating portions of the routing system shown in Fig. 1;

[0018] Fig. 4 is a block diagram illustrating an escape character insertion component consistent with the principles of the present invention;

[0019] Fig. 5 is a diagram conceptually illustrating an implementation of the gap inserter shown in Fig. 4;

[0020] Fig. 6 is a flow chart illustrating methods for operating the gap inserter consistent with principles of the present invention;

[0021] Fig. 7 illustrates exemplary values of a four byte wide data stream; and

[0022] Fig. 8 is a diagram conceptually illustrating an implementation of a byte unpacker.

DETAILED DESCRIPTION

[0023] The following detailed description of the invention refers to the accompanying drawings. The same reference numbers in different drawings identify the same or similar elements. Also, the following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims and equivalents.

[0024] As described herein, a pipelined escape character inserter efficiently inserts escape characters into a continuous multi-byte datapath. The escape character inserter includes stages that insert gaps into the datapath only where necessary. A final stage fills in the gaps with the appropriate escape characters.

SYSTEM DESCRIPTION

[0025] Fig. 2 is a block diagram illustrating an exemplary routing system 200 in which the principles of invention may be implemented. System 200 includes packet forwarding engines (PFEs) 244, 246...248, a switch fabric 250, and a routing engine (RE) 252. System 200 receives a data stream from a physical link, processes the data stream to determine destination information, and transmits the data stream out on a link in accordance with the destination information.

[0026] RE 252 performs high level management functions for system 200. For example, RE 252 communicates with other networks and systems connected to system 200 to exchange information regarding network topology. RE 252 creates routing tables based on network topology information and forwards the routing tables to PFEs 244, 246, and 248. The PFEs use the routing tables to perform route lookup for incoming packets. RE 252 also performs other general control and monitoring functions for system 200.

[0027] PFEs 244, 246, and 248 connect to RE 252 and switch fabric 250. PFEs 244, 246, and 248 receive data at ports on physical links connected to a network, such as a wide area network (WAN). Each physical link could be one of many types of transport media, such as optical fiber or Ethernet cable. The data on the physical link is formatted according to one of several protocols, such as the synchronous optical network (SONET) standard, an asynchronous transfer mode (ATM) technology, or Ethernet.

[0028] PFE 244 will be used to discuss the operations performed by PFEs 244, 246, and 248 consistent with the principles of the invention. PFE 244 processes incoming data by stripping off the data link layer. PFE 244 converts header information from the remaining data into a data structure referred to as a notification.

[0029] For example, in one embodiment, the data remaining after the data link layer is stripped off is packet data. PFE 244 converts the layer 2 (L2) and layer 3 (L3) packet header information included with the packet data into a notification. PFE 244 stores the notification, some control information

regarding the packet, and the packet data in a series of cells. In one embodiment, the notification and the control information are stored in the first two cells of the series of cells.

[0030] PFE 244 performs a route lookup using the notification and the routing table from RE 252 to determine destination information. PFE 244 may also further process the notification to perform protocol-specific functions, policing, and accounting, and might even modify the notification to form a new notification.

[0031] If the destination indicates that the packet should be sent out on a physical link connected to PFE 244, then PFE 244 retrieves the cells for the packet, converts the notification or new notification into header information, forms a packet using the packet data from the cells and the header information, and transmits the packet from the port associated with the physical link.

[0032] If the destination indicates that the packet should be sent to another PFE via switch fabric 250, then PFE 244 retrieves the cells for the packet, modifies the first two cells with the new notification and new control information, if necessary, and sends the cells to the other PFE via switch fabric 250. Before transmitting the cells over switch fabric 250, PFE 244 appends a sequence number to each cell, which allows the receiving PFE to reconstruct the order of the transmitted cells. Additionally, the receiving PFE uses the notification to form a packet using the packet data from the cells, and sends the packet out on the port associated with the appropriate physical link of the receiving PFE.

[0033] In summary, RE 252, PFEs 244, 246, and 248, and switch fabric 250 perform routing based on packet-level processing. The PFEs store each packet using cells while performing a route lookup using a notification, which is based on packet header information. A packet might be received on one PFE and go back out to the network on the same PFE, or be sent through switch fabric 250 to be sent out to the network on a different PFE.

[0034] Fig. 3 is a detailed block diagram illustrating portions of routing system 200. PFEs 244, 246, and 248 connect to one another through switch fabric 250. Each of the PFEs may include one or more physical interface cards (PICs) 301-302 and flexible port concentrators (FPCs) 305.

[0035] PIC 301 may include interfacing, processing, and memory elements necessary to transmit data between a WAN physical link and FPC 305. Different PICs are designed to handle different types of WAN physical links. For example, PIC 301 may be an interface for an optical link while PIC 302 may be an interface for an Ethernet link. Although Fig. 3 shows two PICs connected to the FPCs, in other embodiments consistent with principles of the invention there can be more or fewer PICs connected to an FPC. PICs 301 and 302 communicate with FPC 305 through PIC communication component 307. In one embodiment, communication component 307 implements a point-to-point continuous data stream with FPC 305. The data stream may be a multi-byte stream that uses predefined characters to embed control information, such as end-of-packet and errored packet information. Escape characters may be used to transmit data that has the same value as the control characters.

[0036] FPCs, such as FPC 305, handle packet transfers to and from PICs 301 and 302, and switch fabric 250. For each packet it handles, FPC 305 performs the previously-described route lookup function.

[0037] Fig. 4 is a block diagram illustrating an escape character insertion component 400, used in inserting escape characters in a data stream. Escape character insertion component 400 may be implemented within communication component 307. In alternative embodiments, escape character insertion component 400 may be implemented externally to communication component 307. For example, PICs 301 and 302 may receive and pass an escaped data stream from the external WANs to FPC 305.

[0038] Escape character insertion component 400 includes a first queue 401, such as a first-in-first-out (FIFO) queue, a gap inserter 402, a cyclic-redundancy-check (CRC) component 406, a second FIFO queue 403, a character specific gap inserter 404, an expansion component 407, and an escape character inserter 405. FIFO 401 receives NB byte wide blocks of data (e.g., NB equals 32 for a 32 byte wide continuous stream of data) from the data stream and buffers the incoming data stream for gap inserter 402. In general, gap inserter 402 passes the received data stream to FIFO 403 with an initial set of gaps included in data blocks so that end-of-packet and CRC information can be inserted.

[0039] More specifically, gap inserter 402 examines its input stream from FIFO 401 and inserts gaps in the stream when it detects an end of a packet. In one implementation, for normal packets, gap inserter 402 inserts either three

or five bytes depending on the type of packet. If the packet is a packet that uses a two-byte CRC, gap inserter 402 inserts a three-byte gap (one for an end-of-packet control character and two for the CRC). Otherwise, if the packet is a packet that uses a four-byte CRC, gap inserter 402 inserts a five-byte gap (one for the end-of-packet control character and four for the CRC). In either case, if the packet includes an error, gap inserter 402 adds an additional byte to the gap for an errored-packet control character.

[0040] CRC 406 calculates the CRC value for the packets and inserts it into the gaps created by gap inserter 402. The packet streams, with their inserted CRC information, are then transmitted through FIFO 403 to character specific gap inserter 404. If character specific gap inserter 404 is not able to handle the bytes from FIFO 403 fast enough such that FIFO 403 begins to fill-up, FIFO 403 may signal FIFO 401 to pause its data transfer to gap inserter 402.

[0041] Fig. 5 is a diagram conceptually illustrating an implementation of character specific gap inserter 404, which inserts gaps based on control characters in the data stream. Input interface 501 receives the NB wide block of data in the stream from FIFO 403. The data paths in the data block are labeled as paths one through path NB. Input interface 501 transmits the received bytes (also called characters) to summers 505-507. More particularly, input interface 501 arranges the bytes in the data stream so that the next byte in the data stream (i.e., the most significant byte in the data stream) is output to summer 505, the next most significant byte in the data stream is output to

summer 506, and so on, until the least most significant byte in the data stream is output to summer 507.

[0042] Each of summers 505-507 calculates a sum based on one or more input bytes. Specifically, summer 505 calculates a sum based on input byte NB and summer 506 calculates a sum based on input byte NB and input byte NB-1. The last summer, summer 507, calculates a sum based on all of input bytes one through NB. Summers 505-507 assign each input byte a value of two if the byte is content data that corresponds to a control character and a value of one otherwise. Stated more formally, summers 505-507 each calculate the sum:

$$\sum_{i} B(i)$$

where *i* indexes the input bytes and *B(i)* is equal to two if the indexed byte corresponds to a control character and one otherwise. For each summer 505-507, *i* runs from the main input byte associated with the summer (i.e., NB for summer 505, NB-1 for summer 506, and 1 for summer 507) to NB.

[0043] The output sums from summers 505-507 are input to comparators 510-512. Each of comparators 510-512 compares its input sum to NB, and outputs a logic one if the sum is less than or equal to NB, and a logic zero otherwise. The results of all the comparators 510-512 are concatenated together by bit mask generator 515 and transferred back to input interface 501 as bit mask 516.

[0044] After receiving bit mask 516, input interface 501 shifts the input data block to output lines 530 based on bit mask 516.

[0045] Summers 505-507 and comparator 510-512, although shown as a logical connection of discrete components, may be implemented as a single combinatorial logic circuit.

[0046] The operation of character specific gap inserter 404 will now be described in more detail with reference to Figs. 6 and 7. Fig. 6 is a flow chart illustrating operation of character specific gap inserter 404 consistent with principles of the invention. Fig. 7 illustrates exemplary values of a four byte wide data stream (NB=4) as it passes through gap inserter 402.

[0047] Character specific gap inserter 404 begins by receiving bytes in the data stream from FIFO 403, (Act 601), and placing the received bytes at the input to summers 505-507. (Act 602). The most significant byte (MSB) (i.e., the next byte in the data stream) received from FIFO 403 is output to the first open one of summers 505-507. Thus, if in the previous shift based on bit mask 516, two of four bytes were shifted to output lines 530, input interface 501 would receive the next two bytes from FIFO 401 and place these two bytes at the inputs of the summer 505-507 that are associated with bytes one and two. The third and fourth bytes would still be present at the input to the third and fourth summers from the previous shift operation.

[0048] In Fig. 7, four exemplary bytes are shown, labeled as bytes "A," the most significant byte, through byte "D", the least significant byte. Assume that byte "B" is an escapable character (indicated by the ^ symbol).

[0049] When all the input bytes are properly placed at the input to summers 505-507, the summers generate their respective sums. (Act 603).

Comparators 510-512 then generate the individual bits for bit mask 516 based on the summed values. (Act 604). In Fig. 7, the sum associated with byte "A" is one, as the first summer sums only the value associated with byte "A". Because byte "A" is not an escape character, its associated value is one. The sum associated with byte "B" is the sum of the value of byte "A" plus the value of byte "B". Because byte "B" is an escape character, its associated value is two, and the final sum is thus equal to three. Similarly, bytes "C" and "D", because they are not escape characters, are associated with the value one, and therefore, their associated sums are 4 and 5, respectively. In this example, NB equals four. Accordingly, the comparison by the first three summers is true, while the comparison by the last summer returns false, thus generating the bit mask "1110".

[0050] After generating the bit mask, input interface 501 shifts the input bytes "upwards" to output lines 530 based on the logic ones in the bit mask. (Act 605). In Fig. 7, the three one bits in the bit mask indicate that input interface 501 is to shift three times, thereby shifting the input bytes up three positions into the four byte output datapath. Accordingly, the output bytes in data path 530 include a blank byte followed by "A", "B", and "C".

[0051] Expansion component 407 receives the blocks of NB wide data streams, including the gaps inserted by gap inserter 402 and character specific gap inserter 404, and rearranges each block such that the gaps are in a position appropriate for the block. For the four bit wide block shown in Fig. 7, for

example, byte unpacker 404 moves the gap to the position immediately before byte "B".

[0052] Fig. 8 is a diagram conceptually illustrating an implementation of expansion component 407. Expansion component 407 includes an input interface 801 that receives the blocks created by character specific gap inserter 404. Input interface 801 transmits the non-blank bytes in a block to summers 805-807. Summers 805-807 operate identically to summers 505-507. Shifter 815 shifts each byte in the input data block based on the sum value calculated for the byte by the corresponding one of shifters 805-807. Specifically, each byte is shifted to an output position in out data lines 830 based on its sum value, where the most significant byte is considered to be located at position 1 and the least significant byte is located at output position NB. In terms of the example shown in Fig. 7, byte "A" has a sum of one and is thus placed in the first position, byte "B" has a sum of three and is thus placed in the third position, and byte "C" has a sum of four and is thus placed in the fourth position. The second position now holds the gap that will be later used to hold the escape character.

[0053] Escape character inserter 405 receives the blocks of data, including the appropriately located gaps, from expansion component 407. At each gap, escape character inserter inserts the predetermined escape byte, such as "h7D." Escape character inserter 405 may also modify the content data byte corresponding to the escape byte by, for example, exclusive-ORing the content data byte with a predetermined constant, such as "h20."

[0054] As described above, a pipelined escape character inserter inserts gaps in a multi-byte wide datapath at positions corresponding to locations at which escape characters are to be inserted. The pipelined escape character inserter efficiently calculates the positions at which to insert the gaps. Accordingly, the complexity of the inserter scales linearly with the width of the data path.

[0055] Although described in the context of a router, concepts consistent with the principles of the invention can be implemented in any system that uses a point-to-point data stream that includes control information embedded in the data stream as control characters. Additionally, although the continuous data streams described above were primarily described using information bytes as the basic unit of transmission, this is an arbitrary unit and could accordingly be modified to other sizes.

[0056] The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention.

[0057] No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article "a" is intended to include one or more items. Where only one item is intended, the term "one" or similar language is used.

[0058] The scope of the invention is defined by the claims and their equivalents.